

DEFENSE INFORMATION INFRASTRUCTURE (DII)
COMMON OPERATING ENVIRONMENT (COE)
PROGRAMMER'S MANUAL

ATTACHMENT 1

Programming Guide Version 2.0.0.1 DRAFT
(Windows NT)

28 June 1996

Prepared for:

Defense Information Systems Agency

Document Control No. DII.2.0.0.1.DRAFT.NT.PG-1

June 14, 1996

**Defense Information Infrastructure (DII)
Common Operating Environment (COE)
Programming Guide**

Version 2.0.0.1

DRAFT

(Windows NT)

Table of Contents

	Preface	1
1.	Writing Programs Using the COE Tools	3
2.	Additional Sources of Information	5
3.	Application Development Overview	7
3.1	Running Your Application	7
4.	Segment Development	9
4.1	Segment Layouts	9
4.2	COE Tools Overview	9
4.2.1	Running the COE Tools From the Command Line	10
4.2.2	COE Runtime Tools	11
4.2.3	COE Developer Tools	11
4.3	Building Your Segment	11
4.3.1	Identifying and Creating Required Subdirectories	11
4.3.2	Creating and Modifying Required Segment Descriptor Files	12
4.3.3	Installing a Segment	12
4.4	Customizing Your Segment	13
4.4.1	Adding Menu Items	13
4.4.2	Adding Icons	18

List of Appendices

A	Sample Segments	21
B	Verifying Segment Syntax and Loading a Segment onto Tape	25
B.1	Running VerifySeg Against the Sample Segment	25
B.2	Running TestInstall Against the Sample Segment	26
B.3	Running MakeInstall Against the Sample Segment	27
C	Installing the Developer's Toolkit	29

List of Figures

Figure 1	Segment Directory Structure	10
Figure 2	Segment Descriptor Files	12

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

The following conventions have been used in this document:

[HELVETICA FONT]	Used to indicate keys to be pressed. For example, press [RETURN].
Courier Font	Used to indicate entries to be typed at the keyboard, Windows NT commands, titles of windows and dialog boxes, and screen text. For example, choose the Restart Computer option.
“Quotation Marks”	Used to indicate prompts and messages that appear on the screen.
<i>Italics</i>	Used for emphasis.

THIS PAGE INTENTIONALLY LEFT BLANK

Section 1

Writing Programs Using the COE Tools

The *Defense Information Infrastructure (DII) Common Operating Environment (COE) Programming Guide* provides an introduction to the capabilities of the DII COE tools, which consist of a set of runtime tools and a set of developer's tools.

This document has been designed to help developers start using the DII COE tools. It explains the basic use of the tools, regardless of whether they are run from a menu or from the command line.

The document consists of the following sections and appendices:

Section/Appendix	Page
Additional Sources of Information Lists documents that provide more information about the DII COE toolkit.	5
Application Development Overview Describes how to develop an application using the DII COE Application Programmer Interfaces (APIs).	7
Segment Development Discusses the different types of segments and the process of segment creation.	9
Sample Segments Describes how to install the sample segments, which can be used to test segment installation and execution.	21
Verifying Segment Syntax and Loading a Segment onto Tape Provides examples of how to convert a segment to the <i>DII COE Integration and Runtime Specification</i> segment format, verify segment syntax, temporarily install a segment, and load a segment onto an installation tape.	25
Installing the Developer's Toolkit Describes how to load the Developer's Toolkit, which contains the components needed to create segments that use COE components.	29

Descriptions assume familiarity with the C programming language and with the Windows NT development environment.

THIS PAGE INTENTIONALLY LEFT BLANK

Section 2

Additional Sources of Information

Reference the following documents for more information about the DII COE toolkit:

- *DII COE API Reference Guide Version 2.0.0.1 DRAFT*, DII.2.0.0.1.DRAFT.RG-1, Inter-National Research Institute, June 14, 1996
- *DII COE Integration and Runtime Specification Version 2.0*, DII COE I&RTS:Rev 2.0, Inter-National Research Institute, October 23, 1995.

THIS PAGE INTENTIONALLY LEFT BLANK

Section 3

Application Development Overview

3.1 Running Your Application

Your application can be invoked in the DII COE environment in two ways: (1) Running your application from a command shell window or (2) invoking your application from an icon. The easiest way to test your application is to invoke it in a command shell window. This gives you easy access to your application for debugging purposes and allows you to check any diagnostic information your application is generating. Section 4.4, Customizing Your Segment, describes how to set up your application to be invoked as a menu item or as an icon.

THIS PAGE INTENTIONALLY LEFT BLANK

Section 4

Segment Development

The segment is the basic building block of the COE runtime environment. A segment is a collection of one or more Computer Software Configuration Items (CSCIs) that are most conveniently managed as a unit. Segments are generally defined to keep related CSCIs together so functionality may be easily included or excluded. All applications must be put in the DII runtime environment segment format in order to be installed onto a DII COE-compliant machine.

Once an application has been put in the proper segment format, the segment can be installed in a disciplined way through instructions contained in files provided with each segment. These files are called segment descriptor files and are contained in a special subdirectory, which is called the segment descriptor subdirectory (*SegDescrip*). Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting segments to or from the system.

The following section discusses the different types of segments and the process of segment creation. For a more detailed explanation of segments, please refer to the DII COE Integration and Runtime Specification, Chapter 5.0, *Runtime Environment*.

4.1 Segment Layouts

In the DII COE approach, each segment is assigned a unique, self-contained subdirectory. DII COE compliance mandates specific subdirectories and files underneath a segment directory. These subdirectories and files are shown in Figure 1. Six segment types exist: Account Group, COTS, Data, Database, Software, and Patch. The precise subdirectories and files required depend on the segment type. Some of the subdirectories shown in Figure 1 are required only for segment submission and are not delivered to an operational site.

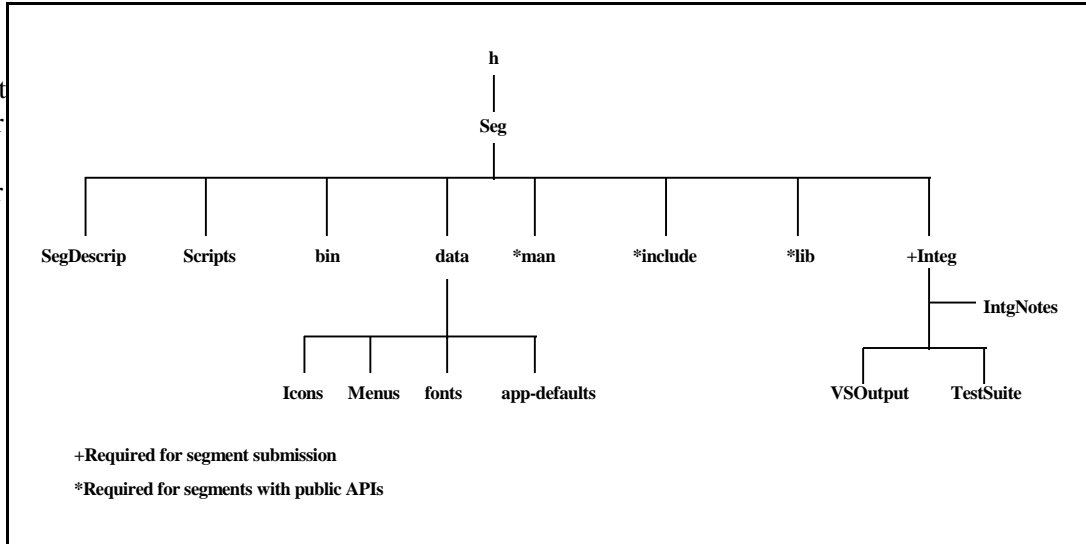
The following runtime subdirectories are normally required: (1) *data*, which is the subdirectory containing static data items, such as menu items, that are unique to the segment, but that will be the same for all users on all workstations, (2) *bin*, which is the directory containing executable programs for the segment, and (3) *SegDescrip*, which is the directory containing segment descriptor files.

The *SegDescrip* directory is required for every segment because it contains the installation instructions for the segment. A segment cannot modify files or resources outside its assigned directory. Files outside a segment's directory are called community files. COE tools coordinate modification of all community files at installation time, while APIs for the segments that own the data are used at runtime. For a detailed explanation on *SegDescrip* files, see the *DII COE Integration and Runtime Specification*, Section 5.5, *Segment Descriptors*.

4.2 COE Tools Overview

The COE tools were constructed to aid developers in the creation and ultimate installation of the DII COE segments. All tools can be run from the command line.

Figure
1.
Segment
Directory
Structure



4.2.1 Running the COE Tools From the Command Line

This section provides a brief overview of running the COE tools from the command line. For detailed information about the COE tools, refer to Appendix C of the *DII COE Integration and Runtime Specification*, the Developer's Toolkit release notes, the on-line help files for the tools, or the help page provided by the tools for the latest information.

When run from the command line, the tools are designed to run interactively and accept one or more command line parameters.

The tools communicate with the outside world in two ways. First, the tools use the exit function to set the DOS status variable. The status return value is set to 0 for normal tool completion or to 255 if an error occurs. A status return value greater than 0 but less than 255 indicates a completion code that is tool specific. The `ERRORLEVEL` function may be used to examine a tool's exit status.

Second, the tools use `stdin` and `stdout` and thus support input and output redirection. Redirecting `stdin` allows the tools to receive input from a file or from another program, while redirecting `stdout` allows the tools to provide output to other programs. [Redirecting `stdin` is not always convenient. The `-R` command line parameter (see below) allows a tool to read input from a response file instead of from `stdin`.] For example, the `COEPrompt` tool displays a message and allows the user to type a response. The user's response, then, is written to `stdout`. The following statement shows how this tool can be used to ask the user to enter the name of a file:

```
COEPrompt "Enter Filename" | MyProg
```

Or, the following statement can be used to write the results to a file:

```
COEPrompt "Enter Filename" > \tmp\tempfile
```

4.2.2 COE Runtime Tools

Reference Appendix C of the *DII COE Integration and Runtime Specification* for a complete description of DII COE runtime tools.

4.2.3 COE Developer Tools

The TestInstall and TestRemove tools must be run as Administrator because they modify files the user may not own. VerifySeg should also be run as Administrator, although it is not mandatory. This tool requires the user to have write permission to the segment against which the tool was executed. Windows NT tools are delivered on a single floppy diskette and may be copied to any directory desired for development.

Reference Appendix C of the *DII COE Integration and Runtime Specification* for a complete description of DII COE developer's tools.

4.3 Building Your Segment

This section describes a process to turn an application into a segment so it can be a part of the COE. As described earlier, a segment is a collection of one or more Computer Software Configuration Items (CSCIs) most conveniently managed as a unit.

A segment must be built in a disciplined way using instructions contained in files provided with each segment. These files are contained in a special directory, called SegDescrip, which is the segment descriptor subdirectory.

4.3.1 Identifying and Creating Required Subdirectories

There are six segment types (i.e., Account Group, COTS, Data, Database, Software, and Patch), and each segment type is assigned its own subdirectory. Precise files depend on the segment type.

The following subdirectories are normally required:

Subdirectory	Description
data	Subdirectory for static data items, such as menu items, that are unique to the segment but that will be the same for all users on all workstations.
bin	Executable programs for the segment. These files can be the result of a compiled program or as a result of shell scripts, depending on the type of segment.
SegDescrip	Subdirectory containing segment descriptor files. This directory is always required for every segment and contains the installation instructions for the segment. A segment is not allowed to directly modify any files for resources it does not "own"; in other words, a segment cannot modify files or resources outside an assigned directory. COE tools coordinate modification of all community files at installation time, while APIs for the segment that owns the data are used at runtime. This subdirectory contains the installation instructions for the segment.

For a detailed explanation of segment directory layout and a description of each SegDescrip file, refer to the *DII COE Integration and Runtime Specification*, Sections 5.0-5.5.

4.3.2 Creating and Modifying Required Segment Descriptor Files

Segment descriptor files are the key to providing seamless and coordinated systems integration across all segments. Refer to Figure 2 to determine which descriptor files are required for the type of segment needed to be installed to the system. For example, the AcctGrp segment requires ReleaseNotes, SegInfo,

SegName, and VERSION descriptor files in the SegDescrip directory, while the Patch segment requires the PostInstall descriptor file in addition to the previously listed files. In DII COE 3.0, some segment information will be contained within individual files while other segment information will be collected into a single file, SegInfo. Table 5-2 in the DII COE *Integration and Runtime Specification* lists the same information as segment descriptor sections within the SegInfo descriptor file.

4.3.3 Installing a Segment

After a segment has been created, follow the procedures below to install the segment.

Running VerifySeg

The software tool VerifySeg must be run during the development phase to ensure segments properly use segment descriptor files. Run the VerifySeg tool whenever a segment is created or modified. VerifySeg must be run for each segment. A Validated file is created when VerifySeg is run without an error. For further information about using VerifySeg, refer to the *DII COE Integration and Runtime Specification*, Appendix C, *COE Tools*.

File	Acct Grp	Aggregate	COE Comp	COTS	Data	DB	S/W Seg	Patch
DEINSTALL	O	O	O	O	O	O	O	O
FileAttribs	O	O	O	O	O	O	O	O
Installed	I	I	I	I	I	I	I	I
PostInstall	O	O	O	O	O	O	O	O
PreInstall	O	O	O	O	O	O	O	O
PreMakeInst	O	O	O	O	O	O	O	O
ReleaseNotes	R	R	R	R	R	R	R	R
SegChecksum	I	I	I	I	I	I	I	I
SegInfo	R	R	R	R	R	R	R	R
SegName	R	R	R	R	R	R	R	R
Validated	I	I	I	I	I	I	I	I
VERSION	R	R	R	R	R	R	R	R

R - Required O - Optional N - Not Applicable I - Created by Integrator or Installation Software

Figure 2. Segment Descriptor Files

Running TestInstall

Executing TestInstall is not a mandatory step in the installation process, but it is recommended. TestInstall simulates an installation on the developer's workstation before actual installation. For further information about using TestInstall, refer to the *DII COE Integration and Runtime Specification*, Appendix C, *COE Tools*.

Running MakeInstall

The MakeInstall tool is used to write one or more segments to an installation media and to package the segment for distribution. MakeInstall checks if VerifySeg has been run successfully on each of the segments and aborts with an error if it has not. For further information about using MakeInstall, refer to the *DII COE Integration and Runtime Specification*, Appendix C, *COE Tools*.

Running COEInstaller

The COEInstaller tool installs a segment from floppy disk. For further information about using COEInstaller, refer to the *DII COE Integration and Runtime Specification*, Appendix C, *COE Tools*.

4.4 Customizing Your Segment

Most properly designed segments will not require any extensions to the COE, except for the need to add icons and menu items. Some segments may need to add special extensions. This subsection describes how to add these items.

4.4.1 Adding Menu Items

Menu files are maintained by the COE, but no COE applications read menu files in this release. Nevertheless, user programs may use their own menu files through this feature.

Menu Entry Format

The Menu Description Entry in the `SegInfo` file contains information on all of the menu entries for a particular menu bar. The entry, or set of entries, to be used is passed to the Common Desktop Environment (CDE) either from the command line or from API routines, depending on the menu bar involved and whether the menus described are base menus or are mode dependent.

The menu bar, pull-down menus, and cascade menus, as well as the menu items they contain, are built according to the menu description entries. The format of the entries is in ASCII with colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A # symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the entry and are not processed by the parser.

Valid keywords are: `MODE`, `PDMENU`, `PDMENUEND`, `ITEM`, `PRMENU`, `CASCADE`, `CASCADEEND`, `APPEND`, `APPENDEND`, and `SEPARATOR`. Since the Menu Description Entry is based on your menu design, you might not use all of these keywords. For example, if your menu does not have separator lines, your Menu Description Entry will not contain a `SEPARATOR` keyword.

Each keyword is described in the following paragraphs.

A **PDMENU** line contains the following elements:

PDMENU: name : enable flag : id # :

PDMENU	Keyword that indicates the start of a pull-down menu.
<i>name</i>	Text used to name the menu. The menu name is displayed on the menu bar.
<i>enable flag</i>	An integer value that indicates whether a menu is enabled or disabled. The enable flag is 1 if a menu is enabled or 0 if it is disabled. A disabled menu means that no options under that pull-down menu can be selected.
<i>id#</i>	<p>An optional integer value that provides a unique ID number for the menu. The PDMENU ID value must be unique within the menu description file. An absolute value may be provided. However, the <i>id#</i> field should be left empty so that relative numbering is used by default. This makes the entry easier to read.</p> <p>With relative numbering, an <i>id#</i> of R1 (or leaving the field blank) sets the menu's ID number to 1 plus the <i>id#</i> of the last menu processed. An <i>id#</i> of R2 sets the menu's ID number to 2 plus the <i>id#</i> of the last menu processed.</p>

The following is an example of a PDMENU line:

PDMENU: Map Options : 1 : R1 :

A **PDMENUEND** line contains the following element:

PDMENUEND:

PDMENUEND	Optional keyword that indicates the end of a group of pull-down menu items. If PDMENUEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.
-----------	---

The following is an example of a PDMENUEND line:

PDMENUEND:

An **ITEM** line contains the following elements:

ITEM: name : command : execution type : enable flag : # instances : id# :
check value : security char : autolog flag : print flag : disk flag :

ITEM	Keyword that indicates a menu item description line.
<i>name</i>	Text used to name the menu item. The item name is displayed in the pull-down menu.
<i>command</i>	Program, with space-separated arguments, that is launched if the menu item type is a program. Otherwise, the menu item is called as an application callback. Because callback functions must be linked into the same executable as the menu bar, applications cannot use callbacks when adding items to the system menu bar.

<i>execution type</i>	An integer value that indicates how a command is to be executed, as follows: <ul style="list-style-type: none"> 1 = executable program 2 = void callback function with no parameters (not yet implemented) 3 = Motif callback function (not yet implemented).
<i>enable flag</i>	An integer value that indicates whether a menu item is enabled or disabled. The enable flag is 1 if a menu item is enabled or 0 if it is disabled. A disabled menu item means that the option cannot be selected.
<i># instances</i>	An integer value that is used to set the maximum number of times the item can be simultaneously executed.
<i>id#</i>	An optional integer value that provides a unique ID number for the menu item. Each <code>ITEM id#</code> entry must be unique within a <code>PDMENU</code> listing. (<code>ITEM</code> entries in a <code>PRMENU</code> must be unique within that <code>PRMENU</code> .) Refer to the <code>id#</code> description under the <code>PDMENU</code> keyword listing.
<i>check value</i>	An optional integer value that sets the star and checks annotations of a menu item. The possible values are: <ul style="list-style-type: none"> 0 = no annotation (default) 1 = visible check mark 2 = check mark, but not visible 3 = visible star 4 = star member, but not visible.
<i>security char</i>	An optional character value that is used to determine the lowest security level under which a menu item can be classified. Valid settings are: <ul style="list-style-type: none"> N = No Classification U = Unclassified (default) C = Classified S = Secret T = Top Secret.
<i>autolog flag</i>	An optional character value, T or F, used to indicate if the command should be automatically logged.
<i>print flag</i>	An optional character value, T or F, used to indicate if the command should have a print capability.
<i>disk flag</i>	An optional character value, T or F, used to indicate if the command should have a disk access capability.

The following is an example of an `ITEM` line:

```
ITEM: Netscape : Netscape.. : 1 : 1 : 1 : R1 : 0 : T : F : F : F :
```

A **`PRMENU`** line contains the following elements:

PRMENU: name : enable flag : id# :

PRMENU	Keyword that indicates a pull right menu button. It is used to mark where a cascade menu is to be connected to an upper-level menu.
<i>name</i>	Text used to name the cascade menu to connect to. The PRMENU name is displayed in the pull-down menu.
<i>enable flag</i>	An integer value that indicates whether a cascade menu is enabled or disabled. The enable flag is 1 if a cascade menu is enabled or 0 if it is disabled. A disabled cascade menu means that menu options on the cascade menu cannot be selected.
<i>id#</i>	An optional integer value that provides a unique ID number for the pull right menu. Each PRMENU id# must be unique within a PDMENU listing. Refer to the id# entry under the PDMENU keyword listing.

The following is an example of a PRMENU line:

PRMENU: Software : 1 : R1 :

A **CASCADE** line contains the following element:

CASCADE: name :

CASCADE	Keyword that indicates the start of a cascade menu. The cascade menu connects to the PRMENU entry of the same name.
<i>name</i>	Text used to name a cascade menu. The name is used to attach a cascade menu to a pull right button. This name must be the same as the name field in the PRMENU entry.

The following is an example of a CASCADE line:

CASCADE: Software :

A **CASCADEEND** line contains the following element:

CASCADEEND:

CASCADEEND	Optional keyword that indicates the end of a group of cascade menu items. If CASCADEEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.
------------	--

The following is an example of a CASCADEEND line:

CASCADEEND:

An **APPEND** line contains the following elements:

APPEND: name :

APPEND	Keyword that indicates the start of a group of items to append to an existing menu. The menu will be created if it does not already exist. The group is appended to the
--------	---

PDMENU or CASCADE entry of the same name.

name Text used to select the menu to which a group of items is appended.

The following is an example of an APPEND line:

APPEND: Options :

An **APPENDEND** line contains the following element:

APPENDEND :

APPENDEND Optional keyword that indicates the end of a group of menu items to be appended to an existing menu. If APPENDEND is not used to delimit a group of menu items, the group is presumed to end when the next keyword (other than ITEM or PRMENU) is encountered.

The following is an example of an APPENDEND line:

APPENDEND :

A **SEPARATOR** line contains the following element:

SEPARATOR :

SEPARATOR Optional keyword that indicates that a Motif separator widget is to be placed in a menu at the point where the keyword occurs.

The following is an example of a SEPARATOR line:

SEPARATOR :

Example of Adding a Menu Item

To add menu items, include the `Menus` Descriptor in the `SegInfo` Segment Descriptor file. Specify the `Menu` file you use wish to load and the `Menu` file you wish to update. The `Menu` file you wish to load should be located under `TstSeg/data/Menu`, assuming the segment name is `TstSeg`. This will add the `Test Program` menu item to the `Software` menu under the `SysAdm` account group.

When invoked through the menu item, the program `TSTCOEAskUser_example` will be executed:

```
[Menus]
TstSegMenu:SA_Default.main

TstSegMenu
#-----
# Software Menu Items
#-----
APPEND          :Software
ITEM            :Test Program    :TSTCOEAskUser_example:1:1:1:R1
APPENDEND :
```

Also include the `$SEGMENT` keyword in the `SegName` Segment Descriptor file to specify the affected segment. In this case it is `System Administration`.

```
#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

4.4.2 Adding Icons

Icon Entry Format

The `Icon Description Entry` contains information on all icon-based processes. The entry, or set of entries, to be used is passed to the `Program Manager`.

Icons are built using the icon heading in the `SegInfo` file. The entry is a specially formatted icon description that has colon-separated fields. The colons are used as delimiters, and spaces are allowed in the fields. Each line ends in a colon with no extra data. A `#` symbol in the first column of a line denotes a comment line. Comment entries may be placed anywhere in the file and are not processed by the parser.

The format of the icon entry is as follows:

```
ICON file : affected icon file
```


The affected icon file contains information about both the icon and the executable. The format of the file is as follows:

```
Icon Name : Icon Path : Executable Path : Comments
```

Where `Icon Name` is the title placed below the icon in the Program Manager, `Icon Path` is the full path to the icon image, `Executable Path` is the full path to the executable to be launched by the menu program, and `Comments` is the comment line.

An example of an affected icon file is as follows:

```
Edit Profiles : /h/AcctGrps/bin/EditProf.ico: /h/AcctGrps/SysAdm/bin/EditProf.exe.
```

Icons are added to a Program Manager group named for the account group to which they belong.

Example of Adding an Icon

To add an icon, include the `Icons` Descriptor in the `SegInfo` Segment Descriptor file. Specify the Icon file you wish to load and the Icon file you wish to update. The Icon file you wish to load should be located under `TstSeg\data\Icons`, assuming the segment name is `TstSeg`. This will add the `Test Program` icon to the `SysAdm` account group. When invoked through the icon, the program `TSTCOEAskUser_example` will be executed.

```
[Icons]
TstSegIcon:SA_Default

TstSegIcons
#-----
# Software Icons
#-----
Test Program :TestProgramIcon:TSTCOEAskUser_example
```

Also include the `$SEGMENT` keyword in the `SegName` Segment Descriptor file to specify the affected segment. In this case it is System Administration.

```
SegName

#
# SegName For Test Segment
#
$TYPE:SOFTWARE
$NAME:Test Segment
$PREFIX:TST
$SEGMENT:System Administration:SA:/h/AcctGrps/SysAdm
```

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Sample Segments

The sample segments delivered with DII COE 2.0.0.1 provide a basic template of a typical DII COE segment. These sample segments can be used to test segment installation and execution. The sample segments contain a sample Account Group segment and a sample Software segment. These examples pass the checks performed by the COE tool VerifySeg. The sample segments are distributed on floppy diskettes.

A hard copy of one of the software segments, called `Seg1`, has been included as a basic example of a segment. As shown in Appendix B, Verifying Segment Syntax and Loading a Segment onto Tape, the sample segment will pass the checks performed by the COE tool VerifySeg.

NOTE: If you do not have the Global Command and Control System (GCCS) Account Group installed on your machine, you will receive several warnings indicating that it must be installed before `Seg1` can be loaded.

The sample segment will add the `Seg1 Hello World` icon to the GCCS Account Group in the Program Manager. When invoked through the icon, the program `Seg1_HelloWorld.exe` will be executed.

Refer to Appendix B for instructions on how to validate the `Seg1` segment and load the segment onto a floppy diskette.

The layout of the sample segment is:

```
Seg1
  bin
    Seg1_HelloWorld.exe
  data
    Icons
      TestIcons
  SegDescrip
    DEINSTALL.BAT
    PostInstall.BAT
    ReleaseNotes
    SegInfo
    SegName
    VERSION
```

The SegDescrip files contain the following:

DEINSTALL

```
REM =====
REM
REM DEINSTALL
REM
REM Routine to perform necessary actions when Seg1
REM is deinstalled.
REM
REM =====
```

PostInstall.BAT

```
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM has been loaded.
REM
REM =====
```

ReleaseNotes

This is the Seg1 Test Segment.

SegInfo

```
#=====
#
#   DII Database Admin Segment SegInfo
#   Descriptor file.
#
#=====
[Hardware]
$CPU:PC
$OPSYS:NT
$DISK:500
$MEMORY:100
[Icons]
TestIcons
[Menus]
[Security]
UNCLASS
```

SegName

```
#=====
#
#   DII Seg1 Test Segment
#   Descriptor file.
#
#=====
$TYPE:SOFTWARE
$NAME:Test Segment #1
$PREFIX:Seg1
$SEGMENT:GCCS COE:GCCS:/h/AcctGrps/GCCS
```

VERSION

1.0.0.1:4/24/96

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Verifying Segment Syntax and Loading a Segment onto Tape

This appendix provides examples of how to verify segment syntax, install a segment temporarily, and load a segment onto an installation floppy diskette. The segment verification and loading process involves the following steps:

- STEP 1: Run the VerifySeg tool to validate that the segment conforms to the rules for defining a segment (i.e., to verify the segment syntax).
- STEP 2: Run TestInstall against the sample segment to install the segment temporarily. This step is optional; if you choose not to run TestInstall, proceed to STEP 3.
- STEP 3: Run the MakeInstall tool to load the segment onto floppy disk. After the segment is loaded onto floppy disk, it is ready to be installed using the `DII Installer` icon from the GCCS Program Manager group.

Subsections B.1-B.3 show how to perform these steps against the `Seg1` sample software segment, which is described in Appendix A.

NOTE: In the following subsections, the VerifySeg, TestInstall, and MakeInstall tools are being run against the `Seg1` sample segment. The output of each command will vary depending on the segment being converted. Note the following severity indicators:

(F)	indicates a FATAL ERROR	(D)	indicates a DEBUG statement
(W)	indicates a WARNING	(V)	indicates a VERBOSE statement
(E)	indicates an ERROR	(O)	indicates an ECHO statement

NOTE: In the following subsections, boldface text indicates information that the user must input.

B.1 Running VerifySeg Against the Sample Segment

```
*****
VerifySeg -p \SampleSegs Seg1

Results of verification (/SampleSegs/Seg1) :
Totals
-----
Errors:      0
Warnings:    0
*****
```

B.2 Running TestInstall Against the Sample Segment

```

*****
TestInstall -p \SampleSegs Seg1

*****
TestInstall - Version 1.0.0.4
*****
The following options have been selected:
*****
Print warning messages.
*****
Segments to be TestInstalled:
*****
Segment: seg1 Path: P:\SampleSegs
*****WARNING*****
TestInstall may modify COE files already in use.
This may cause unpredictable results if COE processes are already running.
Make sure no other COE processes are running before using TestInstall.
Do you want to continue with the TestInstall? (y/n): y
Processing seg1

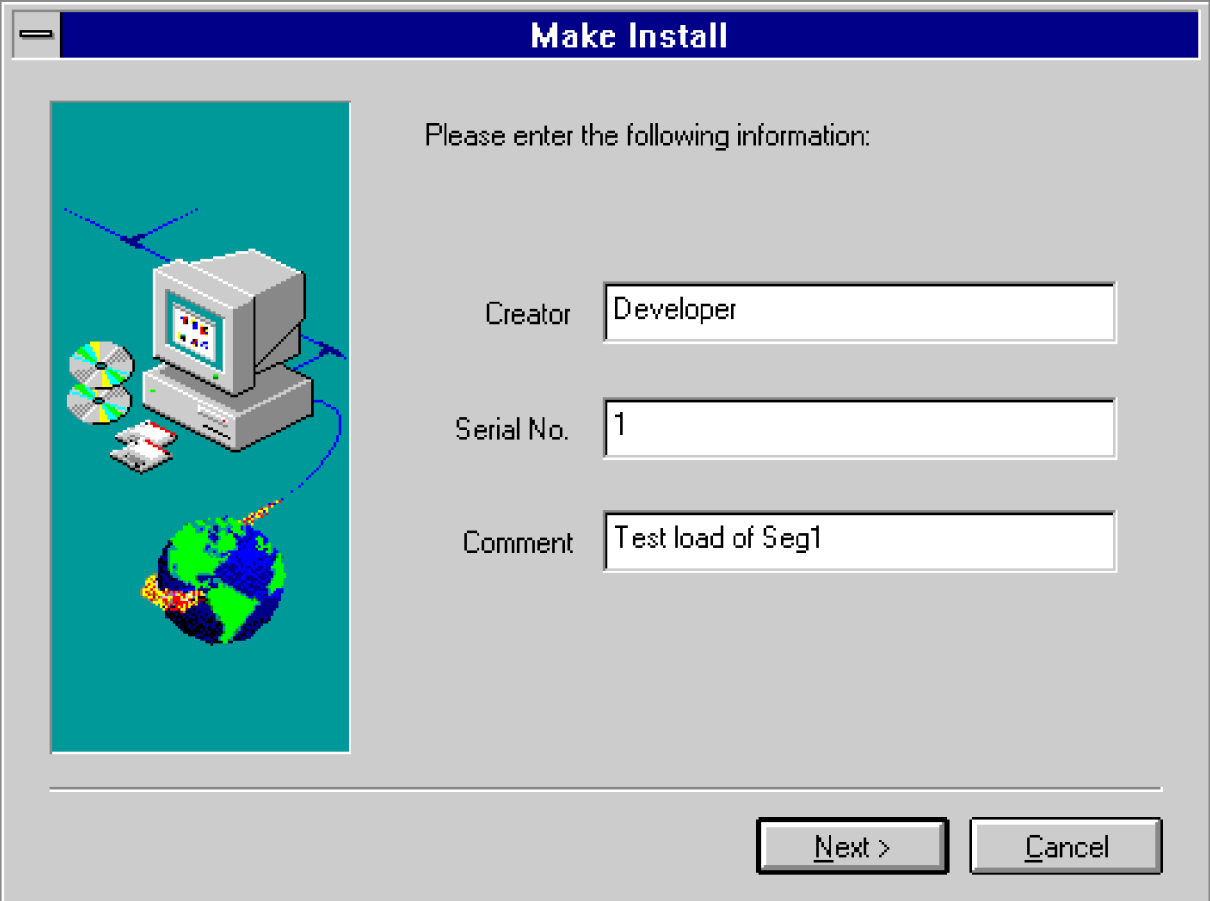
    Successfully ran preprocessor on segment seg1
(W)-----|
    The "Conflicts" data for "P:\nt/seg1" not found
    - Assuming no conflicting segments
No PreInstall script for segment seg1
Do you want to run PostInstall for Segment seg1? (y/n): y
REM =====
REM
REM PostInstall
REM
REM Routine to perform necessary actions after Seg1 Test
REM Segment has been loaded.
REM has been loaded.
REM
REM =====
Successful Installation of seg1
*****

```


B.3 Running MakeInstall Against the Sample Segment

The MakeInstall tool does not generate any command window output. The example below shows the three windows displayed by MakeInstall as it creates a floppy disk. These windows should appear in the order shown.

MakeInstall -p \SampleSegs Seg1




Make Install

Please enter the following information:

Creator:

Serial No.:

Comment:

DISK INFORMATION

Please insert disk 1 to drive A:.

Make Install will erase ALL data from your disk.

Appendix C

Installing the Developer's Toolkit

The Developer's Toolkit contains the components needed to create segments that use DII COE components. The Developer's Toolkit is distributed on floppy diskettes.

Developer's tools are delivered separate from the kernel. Developer's tools can be run from the command line.

Follow the steps below to install and run the DII COE 2.0.0.1 Developer's Toolkit for Windows NT:

STEP 1: Choose a directory on your hard drive into which the tools can be installed.

NOTE: These tools are not location sensitive and can be moved to any directory desired for development ; however, the tools must remain in the same directory structure (i.e., `TOOLS\bin` and `TOOLS\bin\MakeInst`).

STEP 2: Use the Windows File Manager to copy the contents of the `TOOLS` directory from the Developer's Toolkit floppy disk to the desired location. To install the developer's sample segments, use the Windows File Manager to copy the contents of the segments' directories from the sample segments' floppy disk to the desired location.

STEP 3: Use the system control panel to add an entry in your `PATH` environment variable for the `TOOLS\bin` directory.

STEP 4: Create or copy a DII segment onto your hard drive. You may wish to use the GCCS Account Group segment.

STEP 5: Type the following command to run `MakeInstall`:

```
MakeInstall -p [segment directory] [segment name]
```